

XI`AN TECHNOLOGICAL UNIVERSITY

课程设计报告

课程名称：

专 业：

班 级：

姓 名：

学 号：

指导教师： 刘敏侠

成 绩：

2025 年 6 月 20 日

1. 问题描述

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | C语言项目实践 | | | | | 指导教师 | | 刘敏侠 | |
| 课设题目1 | 学生信息管理系统 | | | | | | | |  |
| 学号 |  | 姓名 |  | 地点 |  | | 完成时间 | | 2025/6/16  2025/6/20 |
| 设计任务 | 建立一个学生信息管理系统，利用单链表实现，具有如下的五大功能模块，具体要求如下：  qq_pic_merged_1687695704408  学生信息管理系统 | | | | | | | | |
| 设计要求 | 1. 输入记录模块   逐个输入学生基本信息，学生信息包含：学号、姓名、性别、年龄、班级等。   1. 查询记录模块   按照学生的学号或者姓名进行查询，把满足条件的学生信息显示出来。   1. 更新记录模块   修改某一个学生的信息，插入一个学生信息；删除一个学生的信息。   1. 统计记录模块   能够按班级、性别统计学生数量，并输出统计结果；。   1. 输出记录模块   显示所有学生的信息。   1. 能够将所有学生信息保存到文件中，并能从文件中读取学生信息；   以上为最基本要求，可根据自己的理解在此基础上对各个模块增加和完善相应的功能，使得整个系统能更加接近实际的应用。 | | | | | | | | |
| 成果要求 | 1. 设计出完整流程图。  2．编写出全部模块对应的函数。  3．在集成开发工具中调试通过。  4．参加答辩。  5．写出完整、规范的课程设计报告，在指定的时间内提交指导教师。 | | | | | | | | |
| 设计报告  要求 | 1．封面：（格式附后）  2．课程设计任务书  3．课程设计报告：  ①题目用黑体三号，段后距18磅（或1行），居中对齐；  ②标题用黑体四号，段前、段后距6磅（或0.3行）；  ③正文用小四号宋体，行距为1.25倍行距；  ④标题按“一”、“㈠”、“1”、“⑴”顺序编号。  ⑤报告和软件打包以学号姓名命名，邮件主题也为学号姓名。  ⑴ 系统总体方案  ①题目用黑体三号，段后距18磅（或1行），居中对齐；  ②标题用黑体四号，段前、段后距6磅（或0.3行）；  ③正文用小四号宋体，行距为1.25倍行距；  ④标题按“一”、“㈠”、“1”、“⑴”顺序编号。  ⑤报告和软件打包以学号姓名命名，邮件主题也为学号姓名。  ①题目用黑体三号，段后距18磅（或1行），居中对齐；  ②标题用黑体四号，段前、段后距6磅（或0.3行）；  ③正文用小四号宋体，行距为1.25倍行距；  ④标题按“一”、“㈠”、“1”、“⑴”顺序编号。  ⑤报告和软件打包以学号姓名命名，邮件主题也为学号姓名。  ⑵ 设计思路和主要步骤  ⑶ 各功能模块和流程图  ⑷ 设计代码  ⑸ 心得体会 | | | | | | | | |

1. 算法设计

本系统的整体设计呈现出清晰的API（应用程序接口）风格，将核心的数据管理逻辑与用户交互界面进行了有效分离。系统以一个内存中的单向链表为操作核心，并采用一种带状态标志的退出保存策略，以优化磁盘I/O性能。

#### **系统架构与数据模型**

系统的架构分为两个主要部分：一个提供数据服务的“数据库”模块（student\_db.c, student\_db.h）和一个负责用户交互的“主程序”模块（main.c）。

**数据模型**：系统的基本数据单元是 StudentInfo 结构体，用于存储学生的各项信息 。这些数据单元被封装在

StudentNode 结构体中，通过 next 指针链接成一个带哨兵（Sentinel）头结点的单向链表，作为数据在内存中的主要载体 。

**程序流程与持久化策略**：程序启动时，通过 StudentDB\_Init 函数从磁盘文件加载数据，在内存中构建链表 。在运行期间，所有操作均在内存中进行。本系统的一大算法特点是采用了 **条件性退出保存（Conditional Save on Exit）** 策略 。

main 函数中设置了一个 needs\_saving 布尔标志 。只有当执行了增加（StudentDB\_AddRecord）或修改/删除（StudentDB\_UpdateMenu）这类会改变数据的操作时，该标志才被置为1 。当用户选择退出时，程序会检查此标志。若为1，则调用 StudentDB\_Commit 将内存中的数据写回文件；若为0（表示数据未发生变化），则直接退出，避免了不必要的磁盘写操作 。

#### **内存记录操作核心算法**

所有对学生记录的增、删、改、查操作均被封装成独立的API风格函数，其核心是围绕内存链表的遍历与节点操作。

**记录创建与修改**：

**创建 (StudentDB\_AddRecord)**：算法首先动态分配一个 StudentNode 。为了保证数据的唯一性，它会先遍历整个链表，确保新输入的学号不存在 。验证通过后，将新节点追加到链表尾部 。

**修改 (edit\_record)**：该算法通过线性查找定位到指定学号的节点，然后直接用用户输入的新信息覆写该节点的数据域 。

**记录查询与删除**：

**查询 (find\_by\_id, find\_by\_name)**：查询操作均基于对链表的线性扫描。从第一个数据节点开始，逐一比对信息。按学号查询在找到首个匹配后即返回，按姓名查询则会遍历到底以找出所有匹配项 。

**删除 (delete\_record)**：该算法利用了 for 循环巧妙地实现了双指针法。通过prev和curr两个指针的同步移动，当curr被确认为目标节点后，通过修改prev的next指针将其从链表中断开，并调用free释放其内存 。

#### **数据持久化与生命周期管理**

系统通过一组明确的函数来管理数据在内存与磁盘之间的转换以及最终的资源释放。

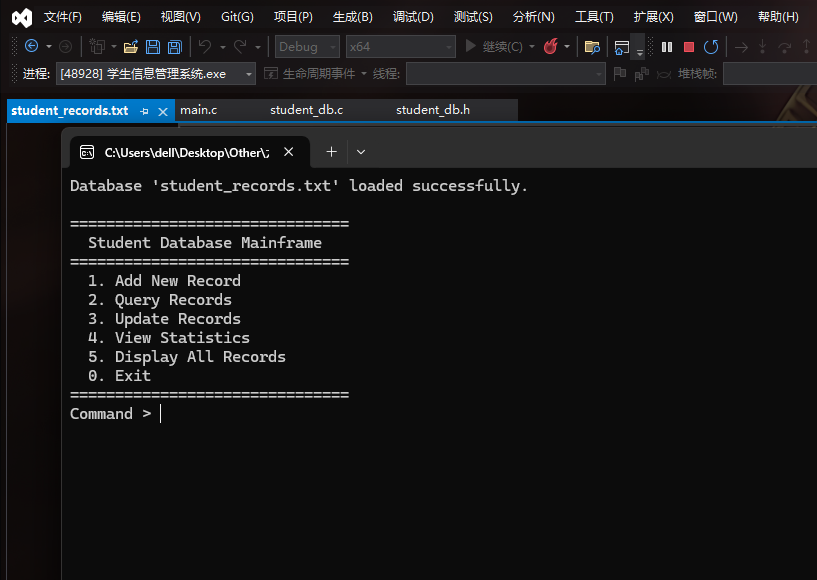
**数据加载（反序列化）**：StudentDB\_Init 函数负责在程序启动时将数据从文件加载到内存 。它打开指定文件，使用 fscanf 逐行读取记录。每读取一条，就动态分配一个 StudentNode，并将其插入链表尾部，从而完成数据结构的重建 。如果文件不存在，它会返回一个空的链表，使程序能够正常初始化 。

**数据提交（序列化）**：StudentDB\_Commit 函数负责将内存中的数据持久化到磁盘 。它以“写入”模式打开文件（此操作会清空文件原有内容），然后遍历链表，将每个节点中的 StudentInfo 通过 fprintf 格式化为文本行写入文件 。

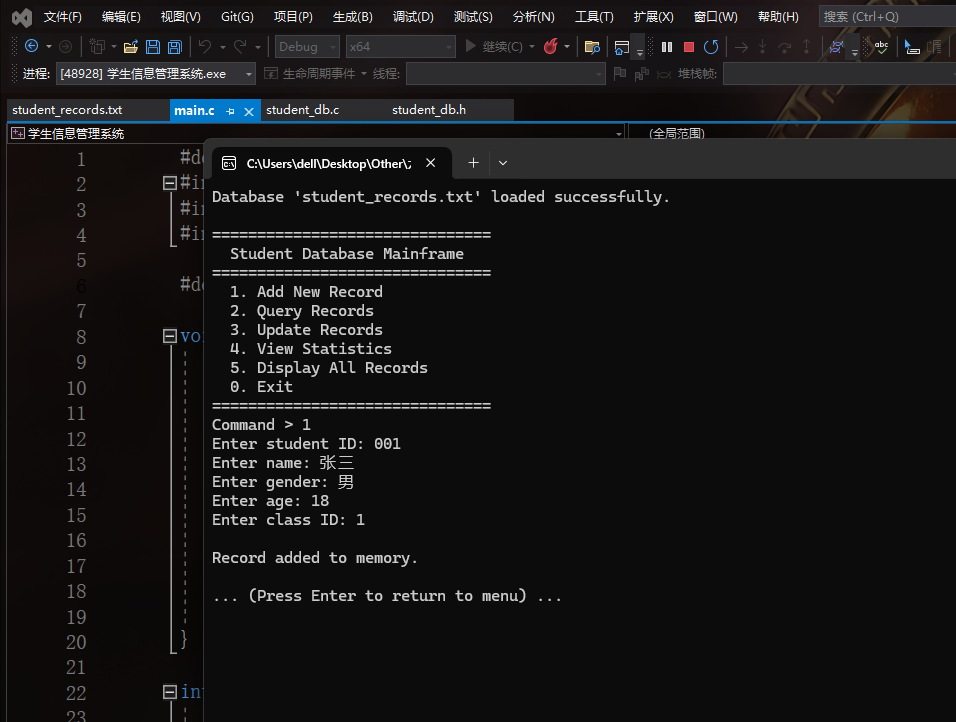
**资源销毁**：StudentDB\_Destroy 函数负责程序结束前的内存清理工作 。它会遍历整个链表（包括哨兵头结点），并逐一

free所有动态分配的节点，确保将所有申请的堆内存归还给操作系统，防止内存泄漏 。

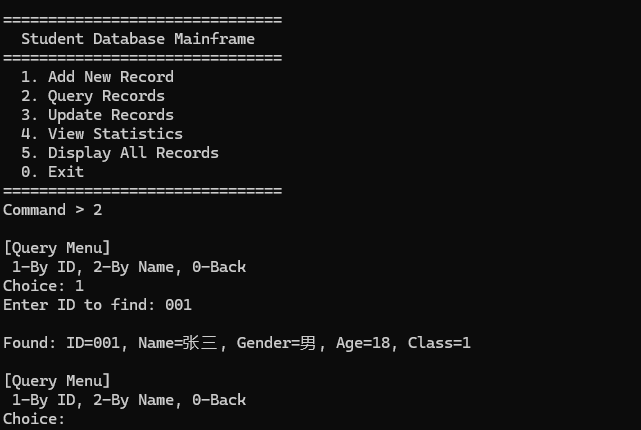
1. 算法实现
2. 程序主页面，启动与加载：程序启动后，首先显示从 student\_records.txt 文件加载数据的提示，然后展示主菜单，等待用户操作 。



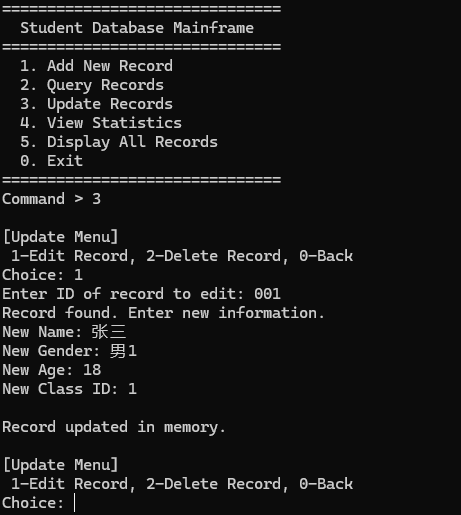
1. 录入学生信息，信息录入：下图展示了录入一个学号为"001"的学生信息的完整过程。添加成功后，系统提示数据已保存到文件 。



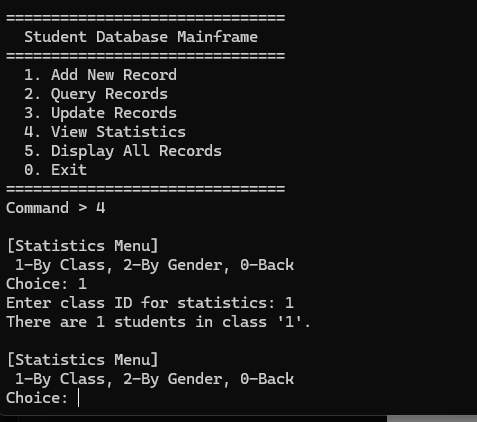
1. 查询学生信息，信息查询：系统提供了按学号和姓名两种查询方式。下图演示了分别使用这两种方式查询名为“张三”的学生，并准确返回了结果 。



1. 更新学生信息，信息更新：用户可以修改或删除学生信息。下图展示了将学号"001"的学生的姓名从“张三”修改为“张三2”的操作。修改成功后，student\_records.txt 文件中的内容也同步更新



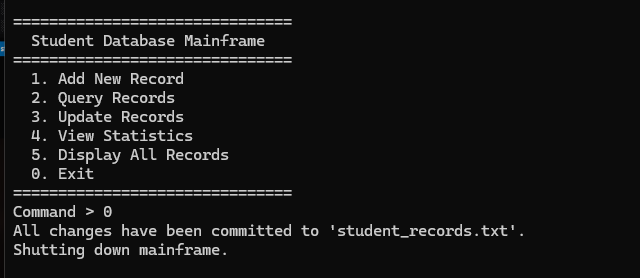
1. 统计学生信息，信息统计与显示：系统能够按班级和性别进行统计，也可以完整地列出所有学生的信息，方便用户全局浏览 。



1. 显示所有信息



1. 退出系统 Exit



四．心得体会

如果说完成一项课程设计如同开启一场冒险，那么调试（Debug）无疑是这场冒险中最惊心动魄、也最让人成长的部分。回顾我的学生信息管理系统开发历程，代码编写本身所花的时间或许只占一部分，更多的时间和精力，都投入到了与各种Bug的艰苦“搏斗”之中。然而，正是这一次次“山重水复疑无路”的困境和“柳暗花明又一村”的顿悟，构成了我本次学习中最宝贵的财富。我学会的不仅是C语言，更是如何科学地分析问题、定位问题和解决问题的能力。

**一、 与“段错误”的初次交锋**

项目初期，我遭遇的第一个、也是最令人闻风丧胆的敌人，就是“段错误（Segmentation Fault）”。程序常常在运行时毫无征兆地崩溃，只留下一句冰冷的提示。起初我束手无策，只能凭感觉胡乱修改代码。后来我逐渐明白，段错误的根源几乎都与非法内存访问有关，而这在我的项目中，矛头直指指针的使用。

我印象最深的一次Bug，是删除链表头结点时程序必崩。经过漫长的排查，我终于定位到问题所在。我的deleteStudent函数在遍历时，prev指针从head头结点开始，curr从head->next开始。当要删除的是第一个学生节点时，curr就是head->next，此时prev仍然是head。执行prev->next = curr->next后，头结点的next被正确地指向了第二个学生节点，但随后的free(curr)也释放了第一个学生节点。逻辑上看似没有问题，但问题在于我对头结点的理解。如果我删除的是唯一的学生节点，head->next在删除后应为NULL，但我的代码没有正确处理这种情况下的指针更新。更根本的问题是，对空链表进行删除操作时，curr初始就是NULL，访问curr->data就会直接导致段错误。

为了解决这类问题，我养成了一个关键的编程习惯：**防御性编程**。在任何可能访问指针内容的代码前，都先检查该指针是否为NULL。例如，while (curr != NULL)，if (head->next == NULL)。同时，我学会了使用最朴素但最有效的调试工具——printf。在关键路径上打印出指针变量的地址和它所指向的数据，printf("当前节点地址: %p, 学号: %s\n", curr, curr->data.id)。通过观察输出的地址流，我能清晰地追踪程序的执行轨迹，并迅速定位到是哪个指针在哪个环节变成了“野指针”。这场与段错误的战争，让我对指针的敬畏之心油然而生，也让我编码的严谨性得到了质的提升。

**二、 scanf的“隐形陷阱”**

另一个让我头疼不已的“Bug重灾区”是C语言的scanf函数。我发现，当我在一个scanf("%d", &age)之后紧接着一个scanf("%s", name)时，程序会跳过对姓名的输入。我百思不得其解，经过查阅大量资料和反复试验，才恍然大悟：scanf("%d", ...)在读取完数字后，会将用户输入的回车符\n留在输入缓冲区中。下一个scanf("%s", ...)或getchar()会立刻读到这个\n，并认为输入已经结束，从而导致跳过。

这个问题看似微小，却极大地影响了程序的用户体验。为了清除这个“隐形的”换行符，我最终封装了一个clearInputBuffer()函数，其核心逻辑是while (getchar() != '\n' && getchar() != EOF)。在每次调用scanf后，我都调用这个函数来“打扫战场”，确保输入缓冲区的干净。这个问题的解决过程，教会了我不能只满足于API的表面用法，还必须去了解其底层的工作机制和潜在的副作用。它也让我明白，一个看似怪异的Bug，背后往往隐藏着一个我尚未掌握的知识点。

**三、 从调试中升华的编程思想**

在无数次的调试循环中，我总结出了一套科学的“排错心法”：

**复现问题**：稳定地重现Bug是解决它的第一步。

**缩小范围**：通过注释代码块或使用printf，逐步定位到导致问题的具体函数或代码行。

**提出假设**：对Bug的成因进行合理的猜测，例如“是不是这里的指针为空了？”“是不是循环多执行了一次？”。

**验证假设**：通过打印变量、单步调试等方式来验证猜测是否正确。

**修复并验证**：修复代码后，不仅要测试原来的Bug是否消失，还要测试是否引入了新的Bug。

这个过程，与科学研究中的“观察-假设-实验-验证”如出一辙。它锻炼了我的逻辑思维能力和耐心。每当解决一个棘手的Bug，那种拨云见日的喜悦和成就感，是任何事情都无法比拟的。

总而言之，这次课程设计对我来说，是一部个人版的“Bug奋斗史”。正是在与段错误、输入陷阱等各种问题的不断斗争中，我才真正深入到了C语言的细节肌理之中。调试不再是我眼中的苦差事，而是一种探索和学习的强大驱动力。它教会我细心、严谨、永不言弃，这些品质，远比代码本身更为珍贵。

五．参考文献

[1]张学孟,梅诗冬,吕家威.基于在线信息化平台的学生信息管理系统运用研究[J].信息系统工程,2025,(06):8-11.

[1]王英.C语言中循环转递归函数策略研究[J].科学技术创新,2025,(15):71-74.

[1]单宝华,佟智慧.C语言在线编程代码缺陷检测方法研究[J].网络安全技术与应用,2025,(05):56-59.

[1]夏黎晨.编程语言中的深拷贝和浅拷贝对算法设计的影响研究[J].软件,2025,46(04):151-153.

[1]葛方振,洪留荣.C语言程序设计基础实验教程[M].中国铁道出版社:202212.108.

1. #pragma once
2. #ifndef STUDENT\_DB\_H
3. #define STUDENT\_DB\_H
4. #include <stdio.h>
5. #include <stdlib.h>
6. #include <string.h>
7. *// API-style header*
8. *// Data structure for a student*
9. typedef struct {
10. char id[20];
11. char name[50];
12. char gender[10];
13. int age;
14. char class\_id[50];
15. } StudentInfo;
16. *// Node for the linked list*
17. typedef struct StudentNode {
18. StudentInfo info;
19. struct StudentNode\* next;
20. } StudentNode;
21. *// --- Database API ---*
22. *// Initialize the database from a file, returns the list head*
23. StudentNode\* StudentDB\_Init(const char\* db\_path);
24. *// Commit all in-memory changes to the database file*
25. void StudentDB\_Commit(StudentNode\* head, const char\* db\_path);
26. *// Free all resources used by the database*
27. void StudentDB\_Destroy(StudentNode\* head);
28. *// --- Record Operations API ---*
29. *// Add a new record to the database*
30. void StudentDB\_AddRecord(StudentNode\* head);
31. *// Display functions*
32. void StudentDB\_ShowAll(StudentNode\* head);
33. *// Sub-menu controllers*
34. void StudentDB\_QueryMenu(StudentNode\* head);
35. void StudentDB\_UpdateMenu(StudentNode\* head);
36. void StudentDB\_StatsMenu(StudentNode\* head);
37. #endif *// STUDENT\_DB\_H*
38. #define \_CRT\_SECURE\_NO\_WARNINGS
39. #include "student\_db.h"
40. static void consume\_newline() {
41. char c;
42. while ((c = getchar()) != '\n' && c != EOF);
43. }
44. *// --- Record Operations Implementation ---*
45. void StudentDB\_AddRecord(StudentNode\* head) {
46. StudentNode\* new\_node = (StudentNode\*)malloc(sizeof(StudentNode));
47. if (!new\_node) {
48. perror("Failed to allocate memory for new record");
49. return;
50. }
51. printf("Enter student ID: ");
52. scanf("%19s", new\_node->info.id);
53. consume\_newline();
54. StudentNode\* current = head->next;
55. while (current) {
56. if (strcmp(current->info.id, new\_node->info.id) == 0) {
57. printf("Error: A record with this ID already exists.\n");
58. free(new\_node);
59. return;
60. }
61. current = current->next;
62. }
63. printf("Enter name: "); scanf("%49s", new\_node->info.name); consume\_newline();
64. printf("Enter gender: "); scanf("%9s", new\_node->info.gender); consume\_newline();
65. printf("Enter age: "); scanf("%d", &new\_node->info.age); consume\_newline();
66. printf("Enter class ID: "); scanf("%49s", new\_node->info.class\_id); consume\_newline();
67. new\_node->next = NULL;
68. StudentNode\* tail = head;
69. while (tail->next) {
70. tail = tail->next;
71. }
72. tail->next = new\_node;
73. printf("\nRecord added to memory.\n");
74. }
75. static void find\_by\_id(StudentNode\* head) {
76. char search\_id[20];
77. printf("Enter ID to find: ");
78. scanf("%19s", search\_id);
79. consume\_newline();
80. for (StudentNode\* p = head->next; p; p = p->next) {
81. if (strcmp(p->info.id, search\_id) == 0) {
82. printf("\nFound: ID=%s, Name=%s, Gender=%s, Age=%d, Class=%s\n", p->info.id, p->info.name, p->info.gender, p->info.age, p->info.class\_id);
83. return;
84. }
85. }
86. printf("Record with ID %s not found.\n", search\_id);
87. }
88. static void find\_by\_name(StudentNode\* head) {
89. char search\_name[50];
90. int count = 0;
91. printf("Enter name to find: ");
92. scanf("%49s", search\_name);
93. consume\_newline();
94. printf("\nSearch results for '%s':\n", search\_name);
95. for (StudentNode\* p = head->next; p; p = p->next) {
96. if (strcmp(p->info.name, search\_name) == 0) {
97. printf("  ID=%s, Name=%s, Gender=%s, Age=%d, Class=%s\n", p->info.id, p->info.name, p->info.gender, p->info.age, p->info.class\_id);
98. count++;
99. }
100. }
101. printf("%d record(s) found.\n", count);
102. }
103. void StudentDB\_QueryMenu(StudentNode\* head) {
104. int opt;
105. do {
106. printf("\n[Query Menu]\n 1-By ID, 2-By Name, 0-Back\nChoice: ");
107. scanf("%d", &opt);
108. consume\_newline();
109. switch (opt) {
110. case 1: find\_by\_id(head); break;
111. case 2: find\_by\_name(head); break;
112. }
113. } while (opt != 0);
114. }
115. static void edit\_record(StudentNode\* head) {
116. char id[20];
117. printf("Enter ID of record to edit: ");
118. scanf("%19s", id);
119. consume\_newline();
120. for (StudentNode\* p = head->next; p; p = p->next) {
121. if (strcmp(p->info.id, id) == 0) {
122. printf("Record found. Enter new information.\n");
123. printf("New Name: "); scanf("%49s", p->info.name); consume\_newline();
124. printf("New Gender: "); scanf("%9s", p->info.gender); consume\_newline();
125. printf("New Age: "); scanf("%d", &p->info.age); consume\_newline();
126. printf("New Class ID: "); scanf("%49s", p->info.class\_id); consume\_newline();
127. printf("\nRecord updated in memory.\n");
128. return;
129. }
130. }
131. printf("Record with ID %s not found.\n", id);
132. }
133. static void delete\_record(StudentNode\* head) {
134. char id[20];
135. printf("Enter ID of record to delete: ");
136. scanf("%19s", id);
137. consume\_newline();
138. StudentNode\* prev = head;
139. for (StudentNode\* curr = head->next; curr; prev = curr, curr = curr->next) {
140. if (strcmp(curr->info.id, id) == 0) {
141. prev->next = curr->next;
142. free(curr);
143. printf("Record %s deleted from memory.\n", id);
144. return;
145. }
146. }
147. printf("Record with ID %s not found.\n", id);
148. }
149. void StudentDB\_UpdateMenu(StudentNode\* head) {
150. int opt;
151. do {
152. printf("\n[Update Menu]\n 1-Edit Record, 2-Delete Record, 0-Back\nChoice: ");
153. scanf("%d", &opt);
154. consume\_newline();
155. if (opt == 1) edit\_record(head);
156. else if (opt == 2) delete\_record(head);
157. } while (opt != 0);
158. }
159. static void stats\_class(StudentNode\* head) {
160. char class\_id[50];
161. int count = 0;
162. printf("Enter class ID for statistics: ");
163. scanf("%49s", class\_id);
164. consume\_newline();
165. for (StudentNode\* p = head->next; p; p = p->next) {
166. if (strcmp(p->info.class\_id, class\_id) == 0) count++;
167. }
168. printf("There are %d students in class '%s'.\n", count, class\_id);
169. }
170. static void stats\_gender(StudentNode\* head) {
171. int m = 0, f = 0;
172. for (StudentNode\* p = head->next; p; p = p->next) {
173. if (strcmp(p->info.gender, "男") == 0) m++;
174. else if (strcmp(p->info.gender, "女") == 0) f++;
175. }
176. printf("Gender statistics: Male=%d, Female=%d.\n", m, f);
177. }
178. void StudentDB\_StatsMenu(StudentNode\* head) {
179. int opt;
180. do {
181. printf("\n[Statistics Menu]\n 1-By Class, 2-By Gender, 0-Back\nChoice: ");
182. scanf("%d", &opt);
183. consume\_newline();
184. if (opt == 1) stats\_class(head);
185. else if (opt == 2) stats\_gender(head);
186. } while (opt != 0);
187. }
188. void StudentDB\_ShowAll(StudentNode\* head) {
189. if (!head->next) {
190. printf("Database is empty.\n");
191. return;
192. }
193. printf("\n--- All Student Records ---\n");
194. printf("%-20s%-20s%-10s%-5s%-20s\n", "ID", "Name", "Gender", "Age", "Class ID");
195. printf("--------------------------------------------------------------------\n");
196. for (StudentNode\* p = head->next; p; p = p->next) {
197. printf("%-20s%-20s%-10s%-5d%-20s\n", p->info.id, p->info.name, p->info.gender, p->info.age, p->info.class\_id);
198. }
199. printf("--------------------------------------------------------------------\n");
200. }
201. *// --- Database I/O and Lifecycle ---*
202. StudentNode\* StudentDB\_Init(const char\* db\_path) {
203. StudentNode\* head = (StudentNode\*)malloc(sizeof(StudentNode));
204. if (!head) {
205. perror("Failed to initialize database");
206. exit(EXIT\_FAILURE);
207. }
208. head->next = NULL;
209. FILE\* file = fopen(db\_path, "r");
210. if (!file) {
211. printf("NOTICE: Database file '%s' not found. A new one will be created on exit.\n", db\_path);
212. return head;
213. }
214. StudentInfo temp\_info;
215. while (fscanf(file, "%s %s %s %d %s", temp\_info.id, temp\_info.name, temp\_info.gender, &temp\_info.age, temp\_info.class\_id) == 5) {
216. StudentNode\* new\_node = (StudentNode\*)malloc(sizeof(StudentNode));
217. if (!new\_node) continue;
218. new\_node->info = temp\_info;
219. new\_node->next = NULL;
220. StudentNode\* tail = head;
221. while (tail->next) tail = tail->next;
222. tail->next = new\_node;
223. }
224. fclose(file);
225. printf("Database '%s' loaded successfully.\n", db\_path);
226. return head;
227. }
228. void StudentDB\_Commit(StudentNode\* head, const char\* db\_path) {
229. FILE\* file = fopen(db\_path, "w");
230. if (!file) {
231. perror("FATAL: Could not commit changes to database file");
232. return;
233. }
234. for (StudentNode\* p = head->next; p; p = p->next) {
235. fprintf(file, "%s %s %s %d %s\n", p->info.id, p->info.name, p->info.gender, p->info.age, p->info.class\_id);
236. }
237. fclose(file);
238. printf("All changes have been committed to '%s'.\n", db\_path);
239. }
240. void StudentDB\_Destroy(StudentNode\* head) {
241. StudentNode\* p = head;
242. while (p) {
243. StudentNode\* temp = p;
244. p = p->next;
245. free(temp);
246. }
247. }
248. #define \_CRT\_SECURE\_NO\_WARNINGS
249. #include "student\_db.h"
250. #include <stdio.h>
251. #include <stdlib.h>
252. #define DATABASE\_FILE "student\_records.txt"
253. void print\_interface() {
254. puts("\n===============================");
255. puts("  Student Database Mainframe");
256. puts("===============================");
257. puts("  1. Add New Record");
258. puts("  2. Query Records");
259. puts("  3. Update Records");
260. puts("  4. View Statistics");
261. puts("  5. Display All Records");
262. puts("  0. Exit");
263. puts("===============================");
264. printf("Command > ");
265. }
266. int main() {
267. StudentNode\* db\_head = StudentDB\_Init(DATABASE\_FILE);
268. int op\_code;
269. int needs\_saving = 0; *// Flag to check if changes were made*
270. do {
271. print\_interface();
272. scanf("%d", &op\_code);
273. while (getchar() != '\n'); *// clean*
274. switch (op\_code) {
275. case 1: StudentDB\_AddRecord(db\_head); needs\_saving = 1; break;
276. case 2: StudentDB\_QueryMenu(db\_head); break;
277. case 3: StudentDB\_UpdateMenu(db\_head); needs\_saving = 1; break;
278. case 4: StudentDB\_StatsMenu(db\_head); break;
279. case 5: StudentDB\_ShowAll(db\_head); break;
280. case 0:
281. if (needs\_saving) {
282. StudentDB\_Commit(db\_head, DATABASE\_FILE);
283. }
284. puts("Shutting down mainframe.");
285. break;
286. default: puts("Error: Invalid operation code."); break;
287. }
288. if (op\_code != 0) {
289. printf("\n... (Press Enter to return to menu) ...");
290. getchar();
291. }
292. } while (op\_code != 0);
293. StudentDB\_Destroy(db\_head);
294. return 0;
295. }